Journal of Simulation (2015) 9, 287–295


www.palgrave-journals.com/jos/

Open

Multi-criteria genetic algorithm applied to scheduling in multi-cluster environments

E Gabaldon*, JL Lerida, F Guirado and J Planes

Universitat of Lleida, Lleida, Spain

Scheduling and resource allocation to optimize performance criteria in multi-cluster heterogeneous environments is known as an NP-hard problem, not only for the resource heterogeneity, but also for the possibility of applying co-allocation to take advantage of idle resources across clusters. A common practice is to use basic heuristics to attempt to optimize some performance criteria by treating the jobs in the waiting queue individually. More recent works proposed new optimization strategies based on Linear Programming techniques dealing with the scheduling of multiple jobs simultaneously. However, the time cost of these techniques makes them impractical for large-scale environments. Population-based meta-heuristics have proved their effectiveness for finding the optimal schedules in large-scale distributed environments with high resource diversification and large numbers of jobs in the batches. The algorithm proposed in the present work packages the jobs in the batch to obtain better optimization opportunities. It includes a multi-objective function to optimize not only the Makespan of the batches but also the Flowtime, thus ensuring a certain level of QoS from the users' point of view. The algorithm also incorporates heterogeneity and bandwidth awareness issues, and is useful for scheduling jobs in large-scale heterogeneous environments. The proposed meta-heuristic was evaluated with a real workload trace. The results show the effectiveness of the proposed method, providing solutions that improve the performance with respect to other well-known techniques in the literature.
Journal of Simulation (2015) **9**(4), 287–295. doi:10.1057/jos.2014.41; published online 30 January 2015

Keywords: multi-cluster; scheduling; co-allocation; genetic algorithms; multi-criteria

1. Introduction

Multi-cluster environments are usually presented as an alternative to high-performance computing for solving large-scale optimization problems by leveraging the computational resources distributed throughout an organization. These environments are made up of several clusters of computers joined by dedicated interconnection networks (Javadi *et al*, 2006). These environments are distinguished from Grid environments in that the multi-cluster uses a dedicated interconnection network between cluster resources with a known topology and predictable performance characteristics, while in Grid, the computing resources are distributed over multiple organizations interconnected through Internet.

A critical aspect of exploiting the resources in a multi-cluster environment is the use of co-allocation. Co-allocation of job tasks across different clusters enables the execution of applications with more requirements than those available in each single cluster. The reduction of the internal cluster fragmentation thus improves the resource usage and increases job throughput as the applications can start their execution earlier. This situation also reduces job waiting times (Bucur and Epema, 2007; Blanco *et al*, 2010). However, mapping jobs across cluster boundaries can result in rather poor overall performance when co-allocated jobs contend for inter-cluster network bandwidth. Additionally, the heterogeneity of processing and communication resources increases the complexity of the scheduling, turning it into an NP-hard problem (Abawajy and Dandamudi, 2003; Jones *et al*, 2005).

The existing methodologies used to solve this problem can be divided into two different groups: Deterministic Algorithms (DA) and Approximate Algorithms (AA). The former can find good solutions among all the possible ones but do not guarantee that the best or a near optimal solution will be found. These methodologies are faster than traditional exhaustive algorithms but inappropriate for large-scale scheduling problems. The latter employ iterative strategies to find optimal or near optimal solutions. Although they are less efficient than DA, they can find good solutions for large-scale problems in a reasonable time.

Genetic Algorithms are an example of AAs able to find efficient solutions for large and complex problems by simulating the behaviour of nature. In this work, we propose a novel approach based on genetic algorithms for solving the parallel job-scheduling problem with co-allocation in heterogenous multi-cluster environments. Our approach treats the jobs in the waiting queue as a set of work packages to provide better scheduling opportunities. The computational heterogeneity and also the inter-cluster link contention are considered to model the applications execution time more accurately.

The rest of the paper is organized as follows. In section 2 related work is presented. The problem statement is described in Section 3. The proposed Genetic Algorithm and its profiling is elaborated in Section 4. Section 5 shows the experimental results

*Correspondence: E Gabaldon, Department of Computer Science, Universitat of Lleida, Avda. Jaume II, n.69, Lleida 25001, Spain.
E-mail: eloigabal@diei.udl.cat

of the proposed GA heuristics in comparison with others techniques in the literature. Finally, the conclusions are presented in Section 6.

2. Related work

The scheduling strategies have generated great interest in recent years due to the growth of resources in organizations. To take advantage of the increasing computational resources, the co-allocation technique allows pieces of jobs to be distributed across different computing clusters. (Bucur and Epemal, 2007) carried out a performance evaluation of different scheduling strategies using co-allocation based on job queues. Their results show that unrestricted co-allocation is not recommended and that performance is improved by limiting the component size of the co-allocated jobs. Other studies used co-allocation to develop load balancing techniques (Heien *et al*, 2008; Yang *et al*, 2008) or optimize the application execution time by selecting the most suitable resources (Jones *et al*, 2005; Naik *et al*, 2005). These studies share the optimization of a single performance metric, such as the computing capability or the communication links usage, without finding a compromise between these. Lérida *et al* (2013) proposed a new model to estimate the application execution time by considering heterogeneity and availability of both resource processing and communicating, that we take as the basis to produce our execution time estimations.

Traditional scheduling techniques in the literature treat the jobs in the waiting queue individually without considering the remaining jobs (Braun *et al*, 2001), thus limiting the scheduling opportunities for future allocation and decreasing overall system performance Shmueli and Feitelson (2005); Tsafrir *et al*, 2007). Shmueli and Feitelson (2005) proposed a backfilling technique in which later jobs are packaged to fill in holes and increase utilization without delaying the earlier jobs. Tsafrir *et al* proposed a method to select the most suitable jobs to be moved forward based on system-generated response time predictions. These techniques however are based on predetermined order, only moving forward jobs that accomplish specific deadline requirements. Blanco *et al* (2011, 2012) proposed diverse techniques for determining the best scheduling of sets of job packages to minimize their overall execution time, based on a Mixed-Integer programming model. Although these techniques produce very good results, their computational cost makes them impractical for large-scale environments.

Algorithms that generate near-optimal schedules have a high time-cost. Conversely, for any upper limit on time-cost, the quality of the schedule, in general, will also be limited. Together, this suggests a trade-off between performance and time-cost, and as a result, many works in the literature propose *ad-hoc* heuristics able to provide lower time-costs, but that obtain far from optimal solutions (Braun *et al*, 2001; Blanco *et al*, 2012).

The Approximate meta-heuristic algorithms, such as Monte Carlo, Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GA), etc, have been presented as effective schedulers in complex large-scale environments in attempts to obtain better schedules. GAs are well known for their robustness and have been applied successfully to solving scheduling problems in a variety of fields. Zomaya and Teh (2001) used GAs in dynamic load balancing problems. Braun *et al* (2001) compared the efficiency of a simple GA-based scheduler and the MinMin, MinMax, Minimum Completion Time (MTC) algorithms. Gabaldon *et al* (2013) presented a GA-based scheduling meta-heuristic capable of treating a set of jobs but focused only on minimizing the makespan. However, from the user point of view, fluctuations in the waiting times due to alterations in the execution order have a negative impact and do not guarantee any level of QoS. In Carretero and Xhafa (2007), the authors presented an extensive study of GAs for designing efficient Grid schedulers where makespan and flowtime are minimized to include QoS in the solutions, but considering independent jobs without inter-cluster communications.

Taking the previous works into account, in this paper the authors define a new scheduling solution to evaluate complete packages of jobs, able to optimize not only the Makespan of the batches but also the Flowtime, thus providing certain level of QoS from the users point of view. To achieve this goal, being also useful for large-scale heterogeneous environments executing parallel jobs, the proposal incorporates heterogeneity and bandwidth aware issues. Owing to the high complexity of the scheduling problem posed, the authors decided to develop a GA-based solution, which is detailed in the following sections. This was due to the ability of such algorithms to explore the solution space exhaustively in a reasonable execution time.

3. Parallel job execution model

In order to perform efficient scheduling in a heterogeneous multi-cluster environment, our proposal have to deal with two challenges: (i) resource heterogeneity and availability, and (ii) task allocation, since tasks from any job can be assigned to different clusters in a co-allocation process. Note that the final job allocation not only has to consider the execution time, but also the communication necessities and availabilities in order to avoid inter-cluster link saturation, which could have an unpredictably effect on the job performance.

3.1. Problem description

A multi-cluster environment consists of a set of α arbitrary sized clusters with heterogeneous resources. Let $\mathcal{C} = \{C_1, \ldots, C_\alpha\}$ be the set of cluster sites. Each cluster consists of a set of computation nodes. Let $\mathcal{N} = \{N_1^1, \ldots, N_n^\alpha\}$ be the set of nodes (or resources), which are connected to each other by a dedicated link through a central switch. Let $\mathcal{L} = \{L_1, \ldots, L_\alpha\}$ be the inter-cluster links, where L_k is the link between the site C_k and the central switch, and let $\{B^1, \ldots, B^\alpha\}$ be the corresponding maximum bandwidth for every link.

The problem consists of scheduling a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ in the multi-cluster. A job J_i is composed of a fixed number τ_i of collaborative tasks. Each task consists of various processing, communication and synchronization phases, and can only be executed in one node. Given a schedule, job J_i is in node N_r, expressed as $J_i \in N_r$, if there is at least one task of job J_i being executed in node N_r.

In our model, each task uses an all-to-all communication pattern with similar processing and communicating requirements, where all tasks start and finish at the same time, following the Bulk-Synchronous Parallel model (Skillicorn *et al*, 1997). Job assignment is static, avoiding re-allocations while the job is being executed. Additionally, jobs can be co-allocated to different clusters in order to reduce their execution time and the internal cluster fragmentation.

In order to compute the execution time for a job in a heterogeneous multi-cluster environment, based on the model presented in Lérida *et al* (2013), we characterize every job by two factors: the Processing Slowdown (*PS*) and the Communication Slowdown (*CS*). The *PS* for job J_j is obtained from the slowest processing node J_j is assigned to, that is, the allocated node providing the maximum processing slowdown:

$$PS_j = \max_{\forall r : J_j r} \left\{ PS_j^r \right\}, \quad J_j \in \mathcal{J}$$

where PS_j^r is the slowdown of job J_j in node N_r, which is inversely proportional to the node computation power.

The co-allocation of a parallel job J_j consumes a certain amount of bandwidth in each inter-cluster link L_k, which is:

$$B_j^k = \left(t_j^k \cdot B_j \right) \cdot \left(\frac{\tau_j - t_j^k}{\tau_j - 1} \right), \quad J_j \in \mathcal{J},\ C_k \in \mathcal{C}$$

where B_j is the required per-task bandwidth, τ_j is the number of tasks in job J_j, and t_j^k is the number of tasks of job J_j allocated in cluster C_k. The first term in the equation is the total bandwidth consumed by tasks of job J_j in cluster C_k, and the second term is the percentage of communication with other clusters.

Saturation occurs when co-allocated jobs use more bandwidth than available, and jobs sharing the link are penalized by an increment in their communication time. The inter-cluster Saturation Degree SD^k relates the maximum bandwidth B^k of each link L_k with the bandwidth requirements of the allocated parallel jobs:

$$SD^k = \frac{B^k}{\sum_{\forall J_j k} \left(B_j^k \right)}, \quad L_k \in \mathcal{L}$$

where B_j^k is the bandwidth of job J_j when it is in node N_k.

When $SD^k < 1$ the link L_k is saturated, and jobs using the link are delayed, otherwise it is not. Then, the communication slowdown for job J_j and link L_k, which depends on the saturation, is expressed by:

$$CS_j^k = \begin{cases} \left(SD^k \right)^{-1} & \text{when } SD^k < 1 \\ 1 & \text{otherwise} \end{cases} \quad J_j \in \mathcal{J},\ C_k \in \mathcal{C}$$

The communication slowdown for job J_j is the CS_j^k from the most saturated used link, expressed by:

$$CS_j = \max_{\forall N_k : J_j k} \left\{ CS_j^k \right\}, \quad J_j \in \mathcal{J}$$

Finally, the estimated execution time for a parallel job J_j is:

$$T_j^e = Tb_j \cdot tc_j, \quad J_j \in \mathcal{J} \tag{1}$$

where Tb_j is the base time of job J_j in dedicated resources, and tc_j is the time cost factor when a job is allocated. It is assumed that the base-time Tb_j is known from user-supplied information, experimental data, job profiling, etc. Previous authors computed the time cost tc_j from the allocated resources without considering communications (Jones *et al*, 2005) or considered a fixed communications penalty when co-allocation is applied (Ernemann *et al*, 2010). In contrast, we model the time cost based on the heterogeneity of the processing resources selected and the availability of the inter-cluster links used. The time cost for job J_j is expressed by:

$$tc_j = \sigma_j \cdot PS_j + \left(1 - \sigma_j\right) \cdot CS_j, \quad J_j \in \mathcal{J}$$

where PS_j denotes the processing slowdown from the resources, CS_j is the communication slowdown from the inter-cluster links, and σ_j is the portion of the total execution time spent on processing.

The optimization problem to be solved can be stated as the minimization of two parameters in the creation of a job schedule: the makespan and the flowtime. The *makespan* is defined as the elapsed time between submitting the first job until the finalization of the last one. The makespan is:

$$\max_{J_i \in \mathcal{J}}(F_i) - \min_{J_j \in \mathcal{J}}(S_j)$$

F_i being the time in which job J_i finishes, which is defined as $F_i = S_i + T_i^e$, where S_i is the time when the job starts and T_i^e the estimated execution time of the job, defined in (1). The smaller the makespan, the faster the workload completion. The *flowtime* is defined as the sum of the response times for all the jobs in the workload. The response time is the elapsed time from submission *Sb* until job finalization F_j. In other words, the flowtime of a job J_j consists of the waiting time T_j^w and the execution time T_j^e. Thus, the flowtime can be expressed as:

$$\sum_{J_j \in \mathcal{J}} \left(T_j^w + T_j^e \right) = \sum_{J_j \in \mathcal{J}} \left(F_j - Sb \right)$$

This metric is usually considered as a QoS criterion from the user point of view.

4. Genetic algorithm

The scheduling of tasks in a multi-cluster environment is a hard problem, where exact methods for finding the optimal solution are impractical due to the long computation time. A better approximation is to use stochastic algorithms, with a near optimal solution with a short computation time. A Genetic Algorithm is one of these methods and the one chosen in the present work.

A Genetic Algorithm is a stochastic search heuristic used to find nearly optimal solutions through nature-based techniques. It starts by creating an initial population of solutions known as individuals, each one encoded using a chromosome. To create a new generation, four steps are performed: ranking the individuals driven by a fitness function, a ranking-based selection, the crossover and the mutation. The algorithm is motivated by the hope that after several generations, the new population will be better than the previous ones.

In the next subsections, the chromosome definition and the GA core functions are detailed.

4.1. Chromosome encoding

Each individual in the GA population is represented by means of a chromosome. In our design, the chromosome corresponds to a sequence of alleles, and an allele corresponds to an integer in a specific position (locus) in the chromosome, representing the job order and task allocation to the computational nodes.

Example 1 A set of four jobs $J_1, \ldots, J_4$, a set of five nodes $N_1, \ldots, N_5$, a set of four tasks for job J_1 ($T_{11}, T_{12}, T_{13}, T_{14}$), a set of two tasks for job J_2 (T_{21}, T_{22}), a set of one task for job J_3 (T_{31}), and a set of two tasks for job J_4 (T_{41}). In our chromosome example, shown in Figure 1, job J_1 is initially assigned to the nodes $\{N_1, N_2, N_3, N_4\}$, job J_2 to nodes $\{N_1, N_2\}$, job J_3 to node N_5, and job J_4 to node N_5. When multiple jobs are assigned to the same node, the loci where the jobs are in the chromosome determine their execution order. In our example, job J_4 waits in the system queue until node N_5, assigned to job J_3, is free. Figure 2 shows the job scheduling Gantt chart considering the node reservation precedences.

4.2. Initial population

To start the evolutionary process, it is necessary to have an initial population composed of a diverse set of individuals to facilitate a thorough exploration of the search space. We analysed two techniques for creating the initial population to observe the convergence of the GA. In the first one, the individuals were generated randomly without considering the node status, that is, whether the nodes selected were busy or free. In the second, the individuals are randomly generated by selecting resource allocations that avoid inter-cluster link saturation. Initially, the GA creates a random permutation of the jobs. Then, for each job in the chromosome, a job is allocated to the free nodes of a randomly selected cluster. If a job needs more nodes than those

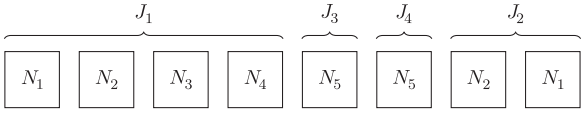

Figure 1 Chromosome design in Example 1.

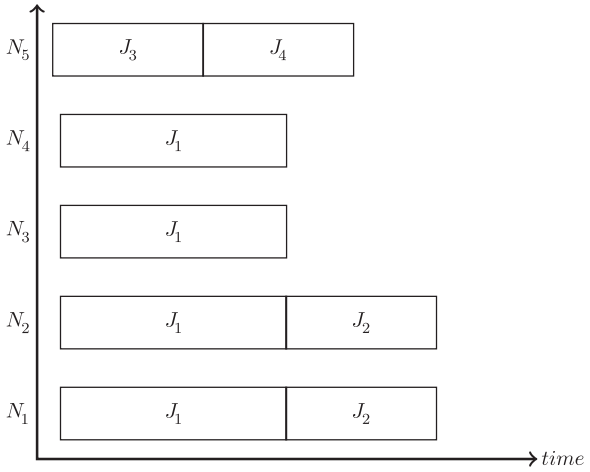

Figure 2 Job schedule in Example 1.

available in the selected cluster, the operation is repeated until all the job tasks are allocated.

If we run out of computational nodes, the GA uses a bandwidth-aware parallel job model to predict the first job to finish and release its allocated nodes for the subsequent jobs. By creating the initial population with this technique, a set of individuals with low inter-cluster communications is provided for the first generation.

An experiment was performed to evaluate the convergence of the two strategies. We experimented with 10 different workloads composed of 200 jobs, a starting population of 50, and the average execution times needed to obtain the results of three executions were computed. Table 1 shows that an initial population created with the second strategy converged faster than the totally random strategy.

4.3. Fitness function definitions

The individuals in the population of each generation are evaluated to obtain the score that determines the quality of the resulting scheduling solutions. The fitness value F for each individual is computed by:

$$F = \alpha \times makespan + (1-\alpha) \times flowtime$$

The α parameter balances the weight of each metric in the score obtained. Besides, the individual assignments that produce network saturation are penalized by avoiding their reproduction in future iterations. For every generation, we record the individual with the minimum F, and the final solution is the best among these.

4.4. Genetic operators

In this paper we consider parallel jobs following the Bulk-Synchronous Parallel model (BSP) (Skillicorn *et al*, 1997). The BSP jobs are made of a fixed number of tasks (not malleable) with similar requirements, and each task comprises various iterations in which computation alternates with communication and synchronization phases. This model is widely used in

Table 1 Number of generations for the GA to converge with the two initial population creation strategies

Workload	GA-Random	GA-Guided
1	600	390
2	220	50
3	300	10
4	190	15
5	300	210
6	50	10
7	350	50
8	1000	810
9	700	100
10	500	50

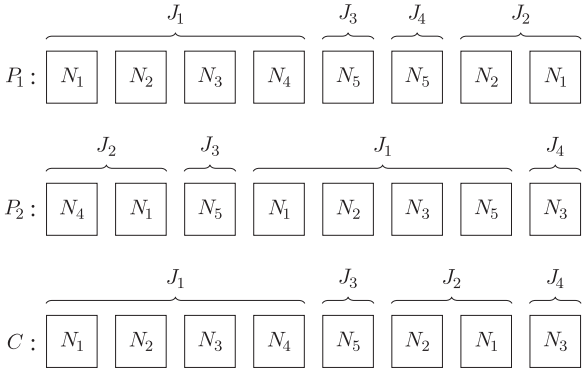

Figure 3 Crossover in Example 2.

scientific computing in such areas as simulation of fluid dynamics (Nibhanupudi *et al*, 1995), combinatorial problems (Abu Salem, 2004), etc. In order to define the suitable scheduling solutions, some constraints must be satisfied for the scheduling solutions:

- All the tasks from the same job must be allocated to different nodes, and each allocated node can only have one task from the same job.
- All tasks from a job start at the same time.
- The computational nodes allocated to any job remain occupied until all the job tasks have finished.

To satisfy these constraints, we define specific genetic operators that ensure the correctness of the solutions: the crossover operator, the selection operator, the mutation operator, and the replacement policy.

The *crossover operator* is the method that uses individuals in the current generation to evolve and create new individuals that will go on to the next generation (offspring). We implemented an ad-hoc crossover function that acts in two stages:

1. *Job-order inheritance*: First, the algorithm compares the parents to be mated. If a job is placed in the same position in both parents, it will remain there. If the job is allocated in different positions in the two parents, it is randomly selected.
2. *Computation node allocation*: Once the job order in the offspring chromosome is defined, the task allocation in the computational nodes is compared. Those that are equal in both parents are copied to the child. The rest are randomly selected.

Example 2 Taking the same set of jobs, tasks and nodes as in Example 1, and given a set of two chromosomes P_1, P_2, the creation of their child C, shown in Figure 3, has the following steps:

1. *Job-order*: Since job J_3 is placed in position 2 in both parents P_1 and P_2, it is also placed at position 2 in child C. The rest of the jobs are placed at random positions in the child: Job J_1 in position 1, job J_2 in position 3 and job J_4 in position 4.
2. *Job allocation*: Since job J_1 is in nodes N_1, N_2, N_3 in both parents P_1 and P_2, these nodes are also selected for the allocation in child C. Node N_4 is randomly selected from the rest of the nodes. Since Job J_3 is in node N_5 in both parents, it is also in this node in child C. Job J_4 is allocated to node N_3 by selecting a random node. Since job J_2 is in node N_1 in both parents P_1 and P_2, such job is also in node N_1 in child C. And finally, node N_2 is randomly selected.

The *selection operator* is used to choose which individuals should mate. The population is arranged with the standard linear ranking selection algorithm—the more suitable they are, the more chances they have of being selected.

Mutation is a common operation used to find new points to evaluate the search space. When a chromosome is chosen for mutation with probability γ, a random choice is made of some genes to be modified. In our case, this process is done in two different phases:

1. *Job ordering mutation*: Any job in the scheduling list can be moved into any other position in the chromosome, allowing different execution orderings to be explored.
2. *Node allocation mutation*: The tasks for the previously selected job can be re-allocated to any other computational node. Note that this mutation must preserve the constraints presented above, which means that mutation is done under supervision.

Our *replacement* policy depends on the selection and mutation schema. Each generation creates a new population based on the selection operator that acts over the previous population. A new individual is obtained from each pair of parent chromosomes. Because the parents are not preserved, the population decreases after each crossover operation. Therefore, to keep the number of chromosomes stable, the GA creates new individuals in each generation.

4.5. Genetic algorithm profiling

The performance of a GA is sensitive to the value of its control parameters, such as *population size, number of iterations* and

frequency of mutation. Furthermore, the proposed fitness function has been defined by the α parameter, which determines how the makespan or flowtime affect the decision process. For this reason, an experimental study was carried out to determine the suitable parameter values to be used in further experimentation.

The parameters were tuned by executing the GA with three different workloads obtained from the real trace HPC2N (Parallel Workloads Archive). Each workload was divided into packets of 20 jobs, based on the results obtained by Gabaldon *et al* (2013), and executed several times to minimize randomization in the results. The multi-cluster system was made up of four clusters with 60 computational nodes with a range of computational capacities and interconnected by a Gigabit network. The multi-criteria fitness function was evaluated in the [20–200] range, with increments of 20 for the *population size* and *number of iterations*, and in the [10–90] range, with increments of 10 for the frequency of mutation. Owing to the differences in the workload job composition, we present the boxplot of the normalized values obtained by the fitness function for each experiment. The time-cost obtained is also evaluated and presented.

The results of the fitness function (primary Y axes) and the time-cost (secondary Y axes) obtained for the population size and number of iterations parameters are shown in Figure 4, and the frequency of mutation parameter, in Figure 5. As can be observed, in general, the higher the value of the evaluated parameter, the better the fitness function obtained, reaching a steady state after a certain threshold. Beside this, the time-cost increases linearly.

With regard to the α parameter, we evaluated the makespan and flowtime for α values ranging from 0 to 1. The α parameter represents the importance of the makespan in the fitness function. Thus, $\alpha = 0$ means that the flowtime completely dominates the fitness function, while $\alpha = 1$ means that only the makespan is considered for evaluating the solution. Any intermediate value represents a combination of both parameters. As Figure 6 shows, increasing α reduces the makespan to 0.6, where a steady state is reached. Figure 6 also shows the results obtained for the flowtime. As can be seen, the higher the makespan weight, the lower the flowtime, until a turning point where the flowtime grows. A more comprehensive study shows that the behaviour of the waiting time and the execution times are inversely proportional. While the waiting time increases with α, the execution time decreases rapidly, the trade-off between the waiting and execution times being at 0.6.

The most suitable value for each parameter is summarized in Table 2 and used in the next section for the performance study of our tuned proposal.

5. Experimental evaluation

The full experimental study was carried out by simulation using the GridSim framework (GridSim simulation framework). This framework was adapted to use our job execution time model and to simulate a dedicated network among the heterogenous multi-cluster environment. We conducted an experimental evaluation with the aim of determining the effectiveness of the proposed GA-based heuristic in a batch multi-cluster scheduler with a real workload trace. The metrics for measuring the effectiveness were the makespan and flowtime. The evaluation was performed with greedy and stochastic heuristics which have been incorporated into the scheduling toolkit of the GridSim framework.

The results were compared with other well-known greedy heuristics for multi-cluster systems that we took as references. These were namely *JPR, CBS* and *METL*. The first, the *JPR*

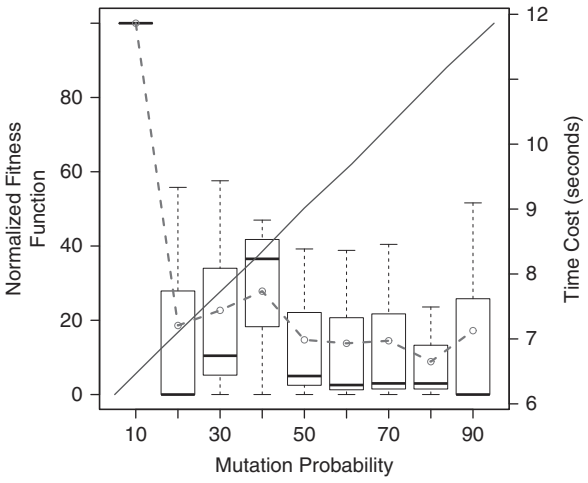

Figure 5 GA profiling: Mutation frequency.

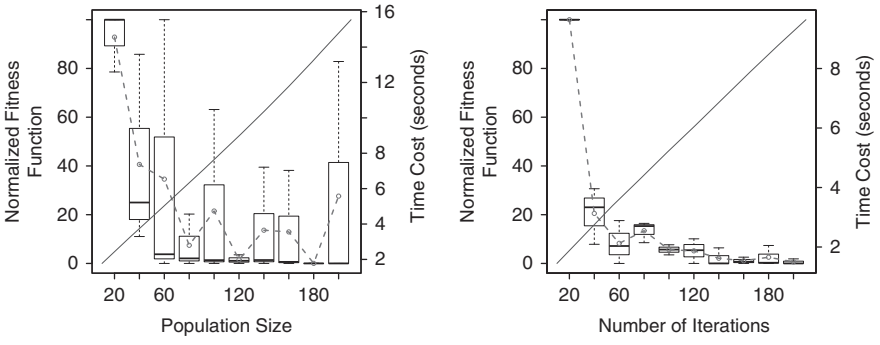

Figure 4 GA profiling: Population size and number of iterations.

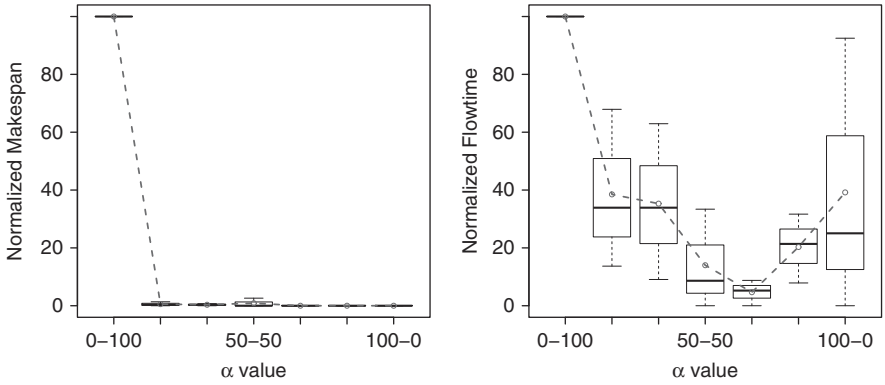

Figure 6 α value evaluation: Makespan and Flowtime.

technique, for *Job Preferences on Resources*, is a variant of the heuristic presented by Naik *et al* (2005), where the tasks are matched with the most powerful available resources to take advantage of the heterogeneity in multi-cluster resources. The second, the *CBS* technique, for *Chunk Big Small*, was presented by Jones *et al* (2005) and tries to allocate a 'large chunk' (75% of the job tasks) to a single cluster in an attempt to avoid inter-cluster link saturation. Finally, the *METL* technique, for *Minimum Execution Time Lost*, is a heuristic presented by Blanco *et al* (2012) which tries to select in each scheduling step the job that minimizes the delay in the execution time with the available resources. This technique considers the saturation of the inter-cluster links in order to model the execution time of the applications.

We also compared our proposal with a GA-based meta-heuristic (GA-MA) and a local search optimization technique (HILL). The former, was proposed by the authors in Gabaldon *et al* (2013), is capable of treating with sets of jobs but only focused on minimizing the makespan. The latter is an implementation of the Hill climbing (HILL) algorithm; it starts with a random solution and attempts to improve such a solution by searching in the neighbourhood. One of the neighbours is chosen by changing the solution by a little. When no better neighbour is found, the last solution is returned.

We assessed the behaviour of our proposals with different workloads obtained from 30 independent real traces in the HPC2N (HPC2N—High Performance Computing Center North) composed of 20 jobs. Most of the jobs in these workloads are computational-intensive with an average execution time of 15 520 s and 90 tasks per job. The multi-cluster system used during the experimentation was made up of four clusters with 60 computational nodes with effective powers of {1000, 1500,-2000, 1500} MIPS, respectively, and interconnected by a Gigabit network. The settings of the GA-MF parameters were those presented in Table 2.

Figure 7 shows the results obtained from evaluating the makespan parameter for each evaluated workload and sorted from the lowest to the highest value for the GA-MF. As can be seen, both the JPR and CBS heuristics gave the worst makespan values. This is because they treat each job individually and attempt to optimize them without taking the global workload requirements into account. Thus, it is possible that the best possible allocation for a job at the moment when it is evaluated may use resources that could improve further job allocations. This problem can be solved by evaluating the full set of jobs in the workload, as METL, HILL, GA-MA and GA-MF do. Note that this is not enough, since Hill Climbing obtains, in average, as good results as CBS and JPR. The figure shows that the results obtained by METL, GA-MA and GA-MF correspond to the best makespan. Evaluating the results obtained in detail, GA-MA and

Table 2 Settings of GA-MF key parameters

Parameter	Value
Num. Iterations	120
Population Size	80
Mutation Frequency	50
α parameter	0.6

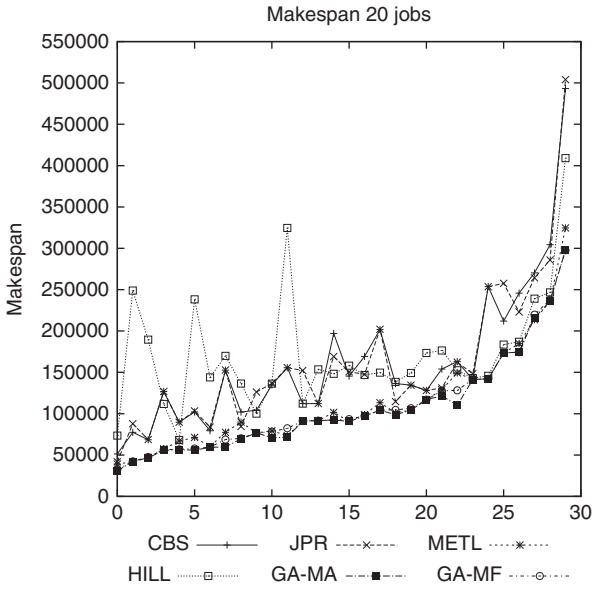

Figure 7 Makespan results for 30 different workloads composed of 20 jobs.

GA-MF have better values than METL, although these are not so significant over this timescale.

The next experimental study was devoted to the flowtime value. The results obtained are shown in Figure 8. As can be seen, the differences between the heuristics and the GA-based meta-heuristics for the flowtime parameter are greater, with HILL, GA-MA and GA-MF giving the best results, especially GA-MF. Although GA-MA and GA-MF obtained similar makespan results, observe how GA-MF is able to find a different job allocation reducing the flowtime dramatically, and thus improving the QoS.

Moreover, since the real workloads are composed of thousands of jobs, we conducted a preliminary study to evaluate how both GA-based meta-heuristics perform in a stressed situation with 25 different workloads, each composed of 2000 jobs. The GA parameters and the number of jobs treated as the entry set were the same as those used in the previous experiment.

Figure 9 shows the results for the makespan metric. As can be observed, both GA-MA and GA-MF obtained the lowest makespan. In this experiment, the differences with the METL heuristic are significant, because GA-MA and GA-MF are able to reduce the makespan for each job package, thus producing a global optimization for the workload.

Figure 10 shows that GA-MA and GA-MF obtained similar results for the flowtime, and this is because a reduced package size does not produce significant benefits with large workloads. Therefore, the job packages have to be larger to take advantage of the capabilities of GA-MF.

We observed that algorithms GA-MA and GA-MF have the best performance in all the experiments, which demonstrate the soundness of choosing such algorithms to solve the multi-cluster scheduling problem. The experiments also show that the more complex the problem (in number of jobs), the larger the performance difference with the other methods.

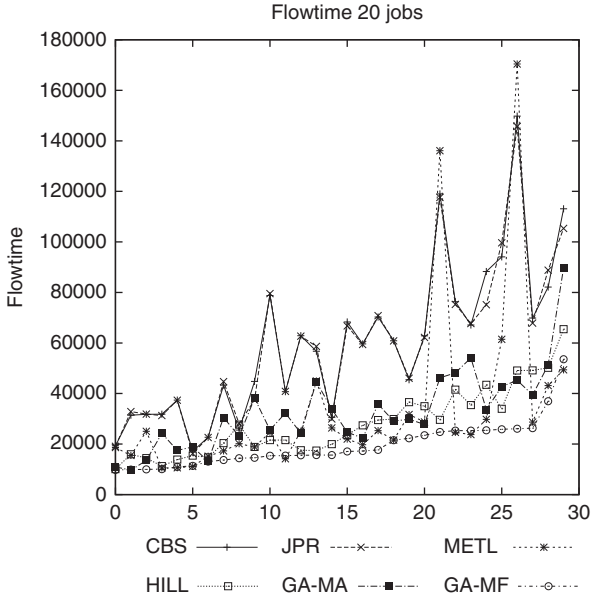

Figure 8 Flowtime results for 30 different workloads composed of 20 jobs.

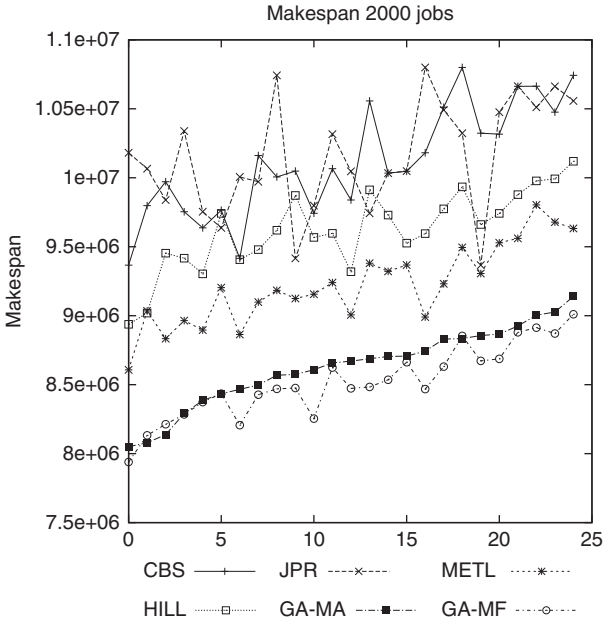

Figure 9 Makespan results for 25 different workloads composed of 2000 jobs.

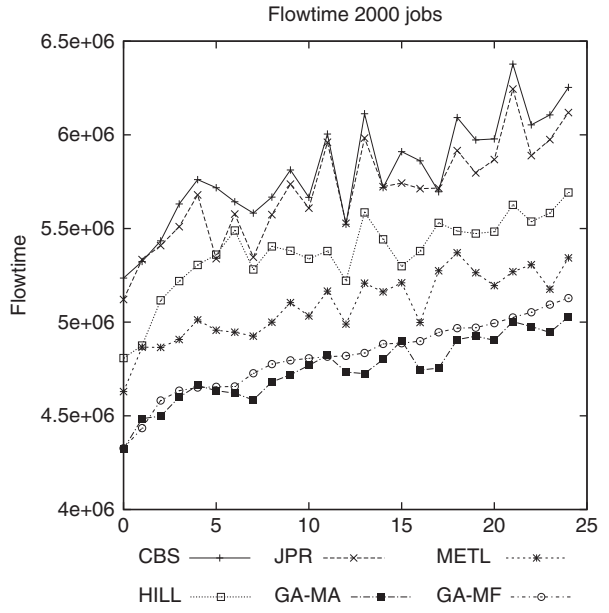

Figure 10 Flowtime results for 25 different workloads composed of 2000 jobs.

6. Conclusions

The simultaneous optimization of certain metrics is a difficult task especially when there is an intrinsic correlation between them. However, it is essential to incorporate multi-criteria optimization to improve the performance and adapt the scheduling techniques to the multiple realities inherent in multi-cluster environments. In this paper, the authors present a novel approach based on a GA-based scheduling meta-heuristic for large-scale

multi-cluster environments applying co-allocation when necessary. The GA-Makespan&Flowtime (GA-MF) meta-heuristic is based on a multi-criteria objective function to minimize both the makespan and the flowtime metrics. The meta-heuristics proposed consider the computational heterogeneity and inter-cluster link contention, addressing the queue as a set of work packages.

The experimental study, based on real traces from the HPC2N workloads, tries to determine the effectiveness compared with our previous GA-MA meta-heuristic and classic heuristics from the literature. The results show that, by using meta-heuristic GA-MF, we can obtain the best flowtime related to the QoS parameter, while also reducing the makespan.

In future work, we aim to reduce the complexity of GA-MF and consider large numbers of jobs together. We also will address the study of new optimization criteria, such as utilization, energy consumption, etc.

Our GA proposal is based on a fitness function obtained by a simulation process; for this reason based on the research done by Nazzal *et al* (2012), we are investigating the introduction of new genetic operators that by using an indifference-zone ranking and selection procedure under common random numbers enables the solution convergence to be speeded up.

Acknowledgements—This work was supported by the Government of Spain under contract TIN2011-28689-C02-02 and the CUR of DIUE of GENCAT and the European Social Fund.

References

Abawajy J and Dandamudi S (2003). Parallel job scheduling on multi-cluster computing systems. In: *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER 2003)*, Hong Kong, China, pp 11–18.

Abu Salem F (2004). A bsp parallel model for the gottfert algorithm over f2. *Parallel Processing and Applied Mathematics* **3019**: 217–224.

Blanco H, Lérida JL, Cores F and Guirado F (2011). Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming. *Journal of Supercomputing* **58**(3): 394–402.

Blanco H, Llados J, Guirado F and Lérida JL (2012). Ordering and allocating parallel jobs on multi-cluster systems. In: *12th International Conference Computational and Mathematical Methods in Science and Engineering* (*CMMSE'12*), Vol. 1. Murcia, Spain, pp 196–206.

Blanco H, Montañola A, Guirado F and Lérida JL (2010). Fairness scheduling for multi-cluster systems based on linear programming. In: *10th International Conference Computational and Mathematical Methods in Science and Engineering (CMMSE'10)*, Vol. 1. Almeria, Spain, pp 227–239.

Braun TD *et al* (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* **61**(6): 810–837.

Bucur AID and Epema DHJ (2007). Scheduling policies for processor coallocation in multicluster systems. *IEEE TPDS* **18**(7): 958–972.

Carretero J and Xhafa F (2007). Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control ICIC* **3**(5): 1053–1071.

Ernemann C, Hamscher V, Schwiegelshohn U, Yahyapour R and Streit A (2010). On advantages of grid computing for parallel job scheduling. *Journal of Supercomputing* **54**(3): 381–399.

Heien EM, Fujimoto N and Hagihara K (2008). Static load distribution for communicative intensive parallel computing in multiclusters. In: *Proceedings of 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*. Toulouse, France.

High Performance Computing Center North http://www.hpc2n.umu.se/, accessed 21 April 2013.

Gabaldon E, Guirado F and Lérida JL (2013). Genetic meta-heuristics for batch scheduling in multi-cluster environments. In: *13th International Conference Computational and Mathematical Methods in Science and Engineering (CMMSE'13)*, Vol. 1. Almeria, Spain, pp 227–239.

Javadi B, Akbari MK and Abawajy JH (2006). A performance model for analysis of heterogeneous multi-cluster systems. *Parallel Computing* **32**(11–12): 831–851.

Jones W, Ligon W, Pang L and Stanzione D (2005). Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *Journal of Supercomputing* **34**(2): 135–163.

Lérida JL, Solsona F, Hernandez P, Gine F, Hanzich M and Conde J (2013). State-based predictions with self-correction on Enterprise Desktop Grid environments. *Journal of Parallel and Distributed Computing* **73**(6): 777–789.

Naik VK, Liu C, Yang L and Wagner J (2005). Online resource matching for heterogeneous grid environments. In: *Proceedings of International Conference on Cluster Computing and the Grid (CCGRID 2005)*, Vol. 2. Cardiff, UK, pp 607–614.

Nazzal D, Mollaghasemi M, Hedlund H and Bozorgi A (2012). Using genetic algorithms and an indifference-zone ranking and selection procedure under common random numbers for simulation optimisation. *Journal of Simulation* **6**(1): 56–66.

Nibhanupudi M, Norton C and Szymanski B (1995). Plasma simulation on networks of workstations using the bulk synchronous parallel model. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'95)*. Athens, Georgia, pp 13–22.

Parallel Workloads Archive http://www.cs.huji.ac.il/labs/parallel/workload, accessed on 21 April 2013.

GridSim simulation framework http://www.buyya.com/gridsim.

Shamueli E and Feitelson DG (2005). Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel and Distributed Computing* **65**(9): 1090–1107.

Skillicorn DB, Hill JMD and McColl WF (1997). *Questions and answers about BSP*. Oxford University Computing Laboratory.

Tsafrir D, Etsion Y and Feitelson DG (2007). Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transaction on Parallel and Distributed Systems* **18**(6): 789–803.

Yang C, Tung H, Chou K and Chu W (2008). Well-balanced allocation strategy for multiple-cluster computing. In: *Proceedings of the 12th International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2008)* Kunming, China.

Zomaya AY and Teh YH (2001). Observations on using genetic algorithms for dynamic load-balancing. *Journal Trans. On Parallel and Distributed* **12**(9): 899–911.

Received 20 February 2014;
accepted 17 December 2014 after one revision